Basi di Dati

Progetto A.A. 2022/2023

TITOLO DEL PROGETTO

0281213

Luca Martorelli

Indice

1. Descrizione del Minimondo

Si vuole realizzare un'applicazione per consentire, ai personal trainer di una palestra, di gestire le schede degli esercizi dei propri clienti. Ciascun utente della palestra è associato ad un personal trainer di riferimento. Questi potrà redigere una scheda di allenamento personalizzata per ciascun utente, scegliendo gli esercizi e i macchinari da utilizzare in un insieme definito dal proprietario della palestra. Per ciascun esercizio, il personal trainer indicherà anche il numero di serie e ripetizioni dell'esercizio.

Periodicamente, quando viene redatta una nuova scheda di esercizi, la scheda precedente viene archiviata. Questa sarà ancora consultabile dagli utenti, ma essi potranno "interagire" solo con quella corrente. Gli utenti della palestra possono infatti effettuare il login nell'applicazione e mostrare quali sono gli esercizi che devono svolgere in una sessione di allenamento. Il sistema mostrerà l'esercizio e la serie corrente, permettendo all'utente di contrassegnare un dato esercizio come completato. L'atleta ha la possibilità di saltare un esercizio e passare al successivo.

I personal trainer possono generare un report che mostra, per tutti i clienti a loro assegnati, quanti sono gli allenamenti sostenuti in un intervallo temporale richiesto, qual è la percentuale di completamento delle schede di allenamento in ogni sessione di allenamento e quanto tempo è durato ciascun allenamento.

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
3	Questi	Il personal trainer	Termine più chiaro.
4	scegliendo gli esercizi e i macchinari da utilizzare	scegliendo gli esercizi da utilizzare da fare	I macchinari sono strumenti con i quali i clienti possono fare degli esercizi, quindi è ridondante andare a specificare sia esercizi che macchinari.
8	Questa	La scheda archiviata	Termine più chiaro.
9	quella	la scheda	Termine più chiaro.
9	infatti	//	Termine inutile nella sintassi della frase.
10	mostrare	vedere	Termine più chiaro.
12	L'atleta	Il cliente	Termine più chiaro.

Specifica disambiguata

Riportare in questo riquadro la specifica di progetto corretta, applicando le disambiguazioni proposte.

Si vuole realizzare un'applicazione per consentire, ai personal trainer di una palestra, di gestire le schede degli esercizi dei propri clienti. Ciascun utente della palestra è associato ad un personal trainer di riferimento. Il personal Trainer potrà redigere una scheda di allenamento personalizzata per ciascun utente, scegliendo gli esercizi da fare in un insieme definito dal proprietario della palestra. Per ciascun esercizio, il personal trainer indicherà anche il numero di serie e ripetizioni dell'esercizio.

Periodicamente, quando viene redatta una nuova scheda di esercizi, la scheda precedente viene archiviata. La scheda archiviata sarà ancora consultabile dagli utenti, ma essi potranno "interagire" solo con la scheda corrente. Gli utenti della palestra possono effettuare il login nell'applicazione e vedere quali sono gli esercizi che devono svolgere in una sessione di allenamento. Il sistema mostrerà l'esercizio e la serie corrente, permettendo all'utente di contrassegnare un dato esercizio come completato. L'atleta ha la possibilità di saltare un esercizio e passare al successivo.

I personal trainer possono generare un report che mostra, per tutti i clienti a loro assegnati, quanti sono gli allenamenti sostenuti in un intervallo temporale richiesto, qual è la percentuale di completamento delle schede di allenamento in ogni sessione di allenamento e quanto tempo è durato ciascun allenamento.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Personal Trainer	Persone che gestiscono schede e report		Clienti, Schede
Clienti	Persone che vogliono usufruire dei servizi della palestra.	Utenti	Personal Trainer, Scheda, Scheda Attiva, Sessione di Allenamento
Scheda	Elenco di esercizi che un Personal Trainer organizza per un cliente.		Esercizi di Scheda, Personal Trainer, Clienti
Schede Attive	Elenco di esercizi che un cliente può contrassegnare come eseguiti o saltati.		Esercizi di Scheda, Personal Trainer, Clienti
Schede Archiviate	Elenco di esercizi che un cliente può solo vedere.		Esercizi di Scheda, Personal Trainer, Clienti
Esercizi	Attività motorie che possono essere presenti in una scheda e svolte da un cliente.		Esercizi di Scheda
Esercizi di Scheda	Attività motorie presenti sulla scheda di un cliente.		Esercizi di sessione, Esercizi, Scheda
Sessione di allenamento	Intervallo temporale nel quale un cliente esegue, o salta, gli esercizi presenti sulla scheda.		Esercizi di Sessione, Clienti.
Esercizi di Sessione	Attività motorie che se presenti sulla scheda di un cliente, quest'ultimo può saltare o eseguire.		Esercizi di Scheda, Sessione di allenamento.

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative a PERSONAL TRAINER
I Personal Trainer sono identificati univocamente da un ID. I Personal Trainer gestiscono le schede dei clienti.
Frasi relative a CLIENTI
I clienti sono identificati univocamente da un Codice Fiscale. È inoltre noto il nome e cognome. Ogni cliente è associato a un Personal Trainer. I clienti possono, inoltre, sia interagire con la scheda attiva, sia visualizzare le possibili schede archiviate.

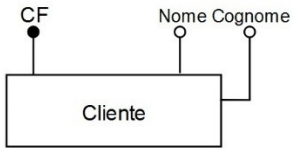
Frasi relative a SCHEDA
Ciascuna scheda è identificata da una data di inizio e dal cliente a cui la scheda è associata. Essa può essere attiva o archiviata.
Frasi relative a SCHEDA ATTIVA
Nel caso di scheda attiva, l'utente può interagirci, scegliendo se eseguire o saltare uno o più esercizi.
Frasi relative a SCHEDE ARCHIVIATE
Nel caso di scheda archiviata, l'utente può solo visualizzare la scheda. Per queste schede sono note anche le date di fine attività (la data in cui la scheda passa da attiva ad archiviata).
Frasi relative a ESERCIZI
Gli esercizi sono identificati dal loro nome.
Frasi relative a SESSIONE DI ALLENAMENTO
Si vuole tener traccia di tutte le sessioni di allenamento fatte da un cliente, in quale data e quanto tempo è durata.

3. Progettazione concettuale

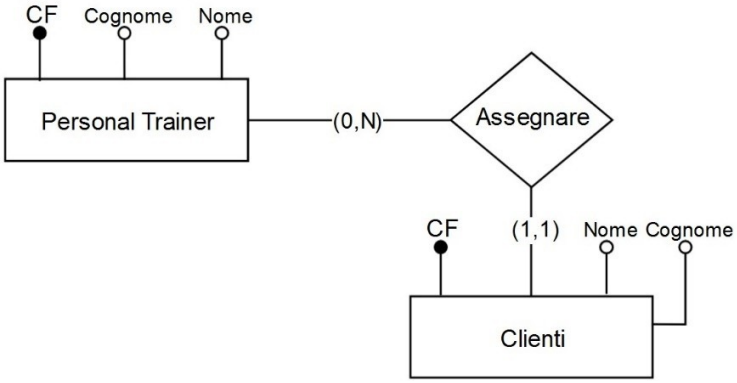
Costruzione dello schema E-R

Organizziamo la progettazione concettuale della nostra base dati adottando la strategia inside-out, che consiste nello sviluppare prima i concetti più importanti (primari) in bottom-up. Gli altri concetti secondari verranno successivamente sviluppati per collegamento ai concetti primari.

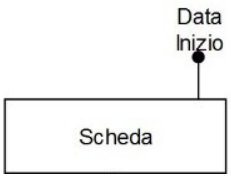
- Il primo concetto del mini-mondo di riferimento è racchiuso nell'entità Cliente. Un Cliente del sistema è identificato dal proprio codice fiscale. Di esso terremo traccia anche del suo nome e cognome. Raggruppando le specifiche che descrivono un cliente si ottiene il seguente schema:



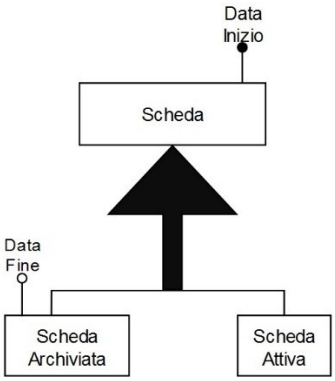
- Introduciamo l'entità "Personal Trainer" all'interno del database, identificata come un "Cliente" attraverso il suo Codice Fiscale. Manteniamo traccia anche del suo nome e cognome. Secondo la specifica, ogni Cliente è associato a un Personal Trainer, quindi è necessario introdurre una relazione tra le due entità descritte in precedenza. La relazione "Assegnare" è di tipo 'uno a molti', poiché i Personal Trainer possono essere associati a più Clienti, ma contemporaneamente ogni cliente fa riferimento a un singolo Personal Trainer.



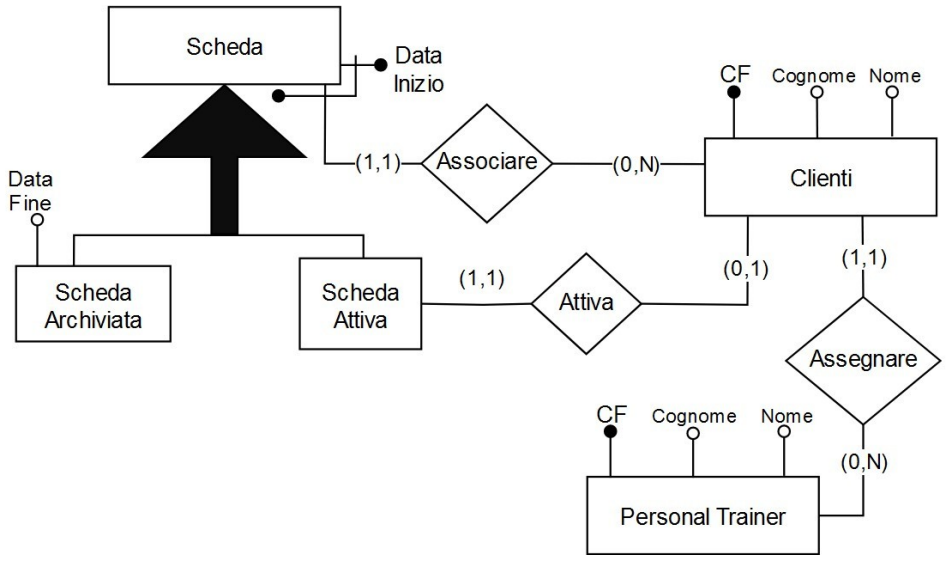
- Successivamente introduciamo il concetto di scheda. Ogni scheda ha una data di inizio identificante.



- La specifica ci fornisce informazioni su come le schede possano essere categorizzate come “schede attive” e “schede archiviate”. Per rappresentare questo concetto, utilizziamo una generalizzazione. Le schede attive saranno completamente accessibili agli utenti, che potranno interagire con esse. D’altra parte, le schede archiviate saranno disponibili solo per la consultazione e presenteranno anche una data di fine attivazione, che rappresenta il giorno in cui tali schede passano da attive ad archiviate.

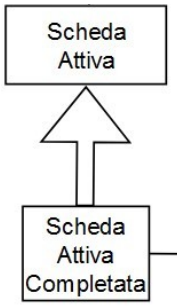


- Tuttavia, sorge un problema: più di una scheda potrebbe avere la stessa data di inizio, rendendo la chiave adottata inizialmente non un vero identificatore. Per risolvere questa situazione, utilizziamo una chiave esterna aggiuntiva per identificare ogni scheda. Oltre alla data di inizio, associamo anche il codice fiscale del cliente a cui la scheda è assegnata, creando così una relazione di entità debole tra “Scheda” e “Cliente”. Questo viene realizzato mediante la relazione “Associare”, con una cardinalità “uno a molti”, che ci consente di tenere traccia di tutte le schede associate a un cliente specifico.

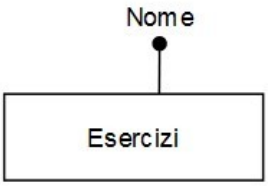


- Durante la fase di progettazione, è fondamentale tenere conto dell’evoluzione dei dati e dei concetti nel tempo. A tal fine, si utilizza il concetto di “pattern di evoluzione di concetto”. Nel contesto

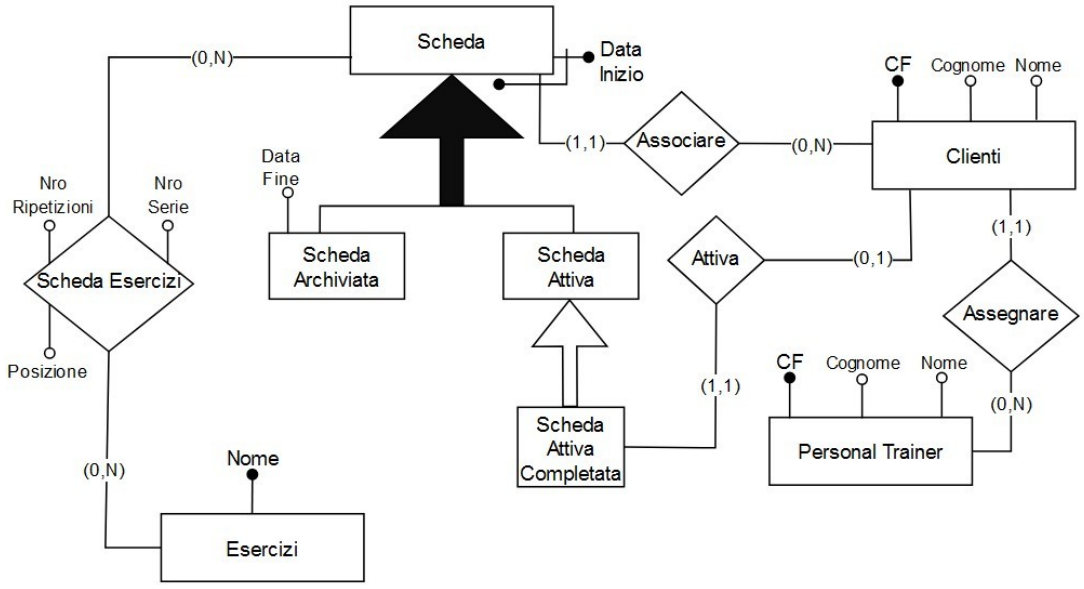
dell'entità "Schede Attive", applicheremo questo pattern per mostrare come inizialmente le schede attive siano "non completate" e successivamente diventino "completate".



- Adesso introduciamo l'entità Esercizi. Essi sono identificati da un nome.

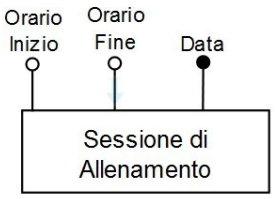


- La specifica ci fornisce una chiara comprensione di come l'entità "Sessione di Allenamento" sia strettamente collegata all'entità "Scheda", descritta in precedenza. Utilizziamo la relazione "Scheda Esercizi" con una cardinalità "molti a molti" per determinare quali esercizi sono inclusi in una determinata scheda di allenamento. Inoltre, questa relazione ci consente di tenere traccia del numero di serie e ripetizioni che ogni cliente deve svolgere per ciascun esercizio presente nella scheda di allenamento; descrivendo, tra l'altro anche l'ordine preferibile, ma non obbligatorio, di esecuzione di ogni esercizio all'interno della scheda tramite l'attributo Posizione. Ovviamente, ogni esercizio avrà una precisa posizione all'interno della scheda, unica tra di loro.



Infine, vogliamo sottolineare l'importanza del concetto di "Sessione di Allenamento". Essa è identificata dalla data in cui viene svolta e contiene informazioni sull'orario di inizio e fine

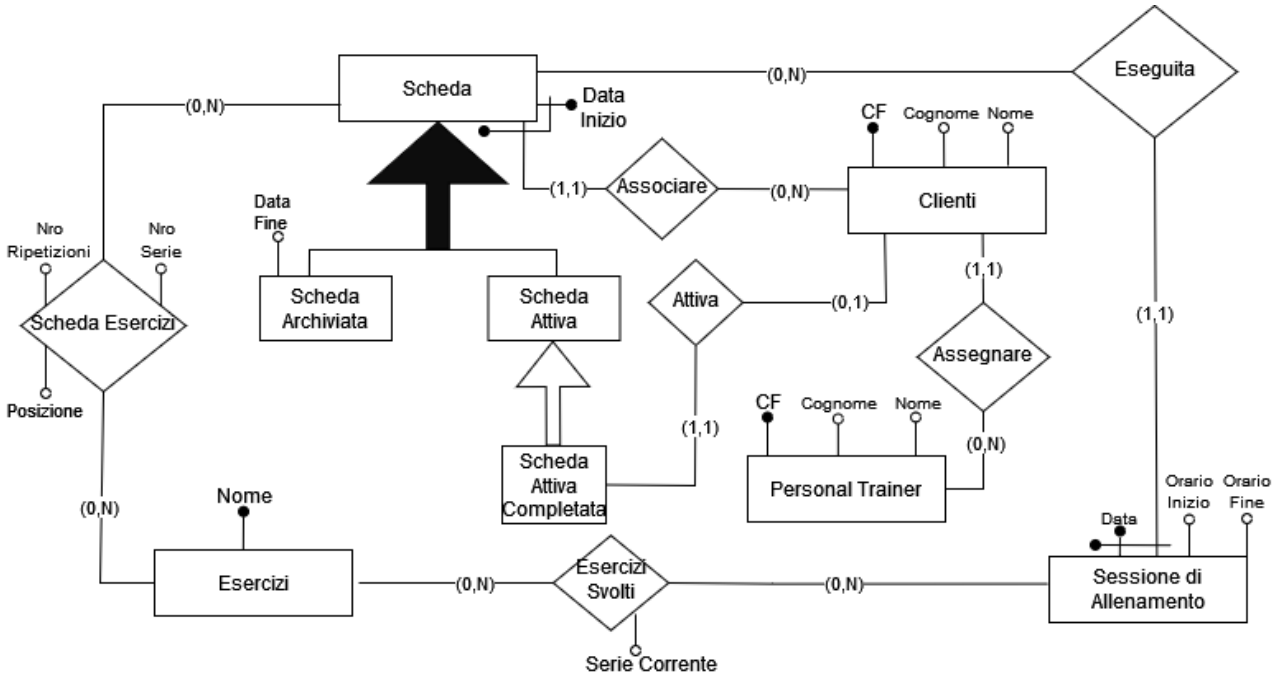
dell'allenamento. Questo concetto è strettamente correlato sia ai Clienti che svolgono le sessioni sia alle Schede di allenamento, che indicano gli esercizi da svolgere durante una Sessione di Allenamento.



- Intendiamo rappresentare la "Sessione di Allenamento" come un'istanza della "Scheda". La "Scheda" raccoglierà tutti i dati sugli esercizi che dovrebbero essere svolti durante una sessione di allenamento, rappresentando così il concetto astratto dell'allenamento. D'altra parte, la "Sessione di Allenamento" rappresenterà i dati sugli esercizi effettivamente svolti dal cliente, catturando quindi il concetto reale.

Considerando quanto appena spiegato, possiamo affermare che la "Sessione di Allenamento" è un'entità debole rispetto alla "Scheda". Inoltre, possiamo collegare la "Sessione di Allenamento" agli esercizi tramite la relazione "Esercizi Svolti". L'associazione "Esercizi Svolti" conterrà anche l'attributo "SerieCorrente", che indica l'ultima serie eseguita di quell'esercizio all'interno della sessione di allenamento.

Integrazione finale



Regole aziendali

1) Un Esercizio, eseguito o saltato, in una Sessione di Allenamento, deve far parte della Scheda Attiva.

2) Il numero di Serie Corrente deve essere minore o uguale di Nro Serie.

3) Una scheda archiviata deve essere completata.

4) Un esercizio, all'interno di una scheda, deve avere una posizione unica.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Cliente	Colui che si allena.	Nome, Cognome	Codice Fiscale
Personal Trainer	Colui che si occupa dell'allenamento dei clienti a lui associati.	Nome, Cognome	Codice Fiscale
Esercizi	Concetto astratto di esercizi.		Nome
Scheda	Elenco di esercizi che un cliente può svolgere in una sessione di allenamento.		Data Inizio, Cliente
Scheda Attiva	Scheda di allenamento che può essere completa o non completa, ed è l'ultima scheda che un PT ha creato per uno dei suoi clienti.		Data Inizio, Cliente
Scheda Attiva Completata	Scheda Attiva che può essere usata dal cliente per allenarsi.		Data Inizio, Cliente
Scheda Archiviata	La Scheda può essere solo letta	Data Fine	Data Inizio, Cliente
Sessione di Allenamento	Intervallo temporale in cui un cliente può svolgere gli esercizi sulla sua scheda attiva	Orario Inizio, Orario Fine	Data, Cliente, DataInizioScheda

4. Progettazione logica

Volume dei dati

Durante questa fase di progettazione, si assume che i dati mantenuti nel database siano relativi agli ultimi 5 anni e che, dopo questo periodo di tempo, il database possa essere ripulito eliminando tutte le schede, sessioni ed esercizi svolti. Effettueremo un'analisi dei volumi, delle frequenze e dei costi del database, tenendo conto del fatto che il numero totale di Clienti e Personal Trainer aumenterà nel tempo. Tuttavia, è importante sottolineare che questa scelta è stata fatta per semplificare il processo di gestione dei dati. Ad esempio, se un cliente si disiscrive dalla palestra e decide di iscriversi nuovamente in futuro, non sarà necessario reinserire tutte le sue informazioni tramite la segreteria. Questa scelta è motivata dal fatto che i dati relativi ai clienti e ai personal trainer rappresentano una parte minima del totale dei dati memorizzati nel database e, pertanto, l'occupazione di memoria associata sarà minima.

1. Si suppone che i dati relativi ai Clienti e ai Personal Trainer che usano l'applicazione siano 1000.

2. Sovrastimiamo, inoltre, che ogni anno in media si registrano 190 clienti, e che di questi, tutti utilizzano in egual modo l'applicazione, in modo costante.

3. Supponiamo che ogni anno si iscrivano in media 10 nuovi Personal Trainer; dunque dopo 5 anni la base di dati conterrà circa 50 PT. Tali numeri sono stati scelti in modo tale da cercare di rendere costante il lavoro che ogni Personal Trainer deve svolgere: assegnando così dopo 5 anni, circa 19 clienti ad ogni PT.

4. Supponiamo che la media di esercizi in una scheda sia di 8. Stimiamo che circa il 75% di questi esercizi verranno effettivamente svolti durante ogni sessione di allenamento. Ogni esercizio prevede in media 3 serie, e supponiamo che vengano eseguite tutte e 3 le serie. Inoltre, ipotizziamo che il cliente visualizzi la sua scheda attiva una volta per ogni sessione di allenamento. Sovrastimiamo inoltre che ogni sessione iniziata, venga anche terminata.

5. Supponiamo inoltre che ci sia una differenza nel numero di clienti che interagiscono con le schede attive rispetto a quelli che consultano le schede archiviate. Stimiamo che, ogni giorno, circa il 55% dei clienti svolga sessioni di allenamento, interagendo con le schede attive, mentre il 10% legga una

scheda archiviata. Di conseguenza, prevediamo che circa il 65% dei clienti presenti nel database effettui il login ogni giorno.

6. Supponiamo inoltre che i Personal Trainer creino in media nuove schede ogni 2 mesi per ogni cliente. Di conseguenza, dopo 5 anni, il numero totale di schede sarà di 17.100. Tra queste, il numero di Schede Attive sarà uguale al numero totale di clienti presenti nel database (950), di cui 713 saranno completate (3/4), mentre le restanti saranno Schede Archiviate.

7. Supponiamo che nel primo anno la palestra abbia a disposizione 25 macchinari. Successivamente, ogni anno la segreteria avrà la possibilità di acquistare 5 nuovi macchinari per la palestra.

8. I clienti hanno la possibilità di allenarsi tutti i giorni, una volta al giorno, dunque stimiamo che ci saranno fra 1 e 7 sessioni di allenamento settimanali (media 4).

9. Assumiamo che ogni volta che un Personal Trainer archivia una scheda per un cliente, ne crei immediatamente un'altra. In media, supponiamo che 3 volte su 4 questa nuova scheda venga completata immediatamente.

Concetto nello schema	Tipo[1]	Volume atteso
Clienti	E	950
Personal Trainer	E	50
Esercizi	E	35
Sessioni di Allenamento	E	547 200
Schede	E	17 100
Schede Attive	E	950
Schede Attive Completate	E	713
Schede Archiviate	E	16 150
Scheda Esercizi	R	85 500
Esercizi Svolti	R	3 283 200
Associare	R	17 100
Attiva	R	17 100
Eseguita	R	547 200
Assegnare	R	950

[1] Indicare con E le entità, con R le relazioni

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
U1	Inizia Sessione	314/giorno
U2	Termina Sessione	314/giorno
U3	Esegui Esercizio	807/giorno
U4	Visualizza Scheda Attiva	314/giorno
U5	Visualizza Scheda Archiviata	57/giorno
U6	Aggiungi Serie Esercizio	5862/giorno
U7	Visualizza Esercizi Mancanti	807/giorno
U8	Visualizzare Serie Mancanti	978/giorno
U9	Crea Scheda	3420/anno ≃ 2/giorno
U10	Archivia Scheda	3420/anno ≃ 2/giorno
U11	Genera Report	360/anno ≃ 1/giorno
U12	Inserisci Esercizio in Scheda	51/giorno
U13	Visualizza Clienti	2/giorno
AM1	Registra Cliente	190/anno
AM2	Registra PT	10/anno
AM3	Visualizza Esercizi	1/settimana ≃ 0.6/giorno
AM4	Aggiungi Esercizi	10/anno
L1	Login	371/giorno

Costo delle operazioni

Si suppone, in questa fase di progettazione, che il costo di scrittura di un dato sia il doppio del costo in lettura.

U1 – Inizia Sessione
- Clienti: 950 (L)
- Attiva: 713 (L)
- Scheda Attiva: 713 (L)
- Eseguita: 547 200 (W)
- Sessione di Allenamento: 547 200 (W)
- Costo Totale: 2 190 939
- Accessi/giorno: (2 190 939) * 314/giorno= 687 954 846

U2 – Termina Sessione
- Clienti: 950 (L)
- Attiva: 713 (L)
- Scheda Attiva: 713 (L)
- Eseguita: 547 200 (W)
- Sessione di Allenamento: 547 200 (W)

Costo Totale: 2 190 939
Accessi/giorno: (2 190 939) * 314/giorno= 687 954 846

U3 – Esegui Esercizio
- Clienti: 950 (L)
- Attiva: 713 (L)
- Scheda Attiva: 713 (L)
- Eseguita: 547 200 (L)
- Sessione di Allenamento: 547 200 (L)
- Esercizi Svolti 3 283 200 (W)

Costo Totale: 7 663 176
Accessi/giorno: (7 663 176) * 807/giorno = 6 184 183 032

U4 – Visualizza Scheda Attiva
- Clienti: 950 (L)
- Attiva: 950 (L)
- Scheda Attiva: 950 (L)

Costo Totale: 2850
Accessi/giorno: (2850) * 314/giorno = 894 900

U5 – Visualizza Schede Archiviate
- Clienti: 950 (L)
- Associare: 17 100 (L)
- Scheda Archiviata: 16 150 (L)
- Scheda Esercizi: 85 500 (L)

Costo Totale: 34 200
Accessi/giorno: (34 200) * 57/giorno = 1 949 400

U6 – Aggiungi Serie Esercizio
- Clienti: 950 (L)
- Attiva: 713 (L)
- Scheda Attiva Completata: 713 (L)
- Eseguita: 547 200 (L)
- Sessione di Allenamento: 547 200 (L)
- Esercizi Svolti 3 283 200 (W)

Costo Totale: 7 663 176
Accessi/giorno: (7 663 176) * 5862/giorno = 44 921 537 712

U7 – Visualizza Esercizi Mancanti
- Clienti: 950 (L)
- Attiva: 713 (L)
- Scheda Attiva: 713 (L)
- Eseguita: 547 200 (L)
- Sessione di Allenamento: 547 200 (L)
- Esercizi Svolti 3 283 200 (L)
- Scheda Esercizi 85 500 (L)

Costo Totale: 4 465 476
Accessi/giorno: (4 465 476) * 807/giorno = 3 603 639 132

U8 – Visualizzare Serie Mancanti

- Clienti: 950 (L)
- Attiva: 713 (L)
- Scheda Attiva Completa: 713 (L)
- Eseguita: 547 200 (L)
- Sessione di Allenamento: 547 200 (L)
- Esercizi Svolti 3 283 200 (L)
- Scheda Esercizi 85 500 (L)

Costo Totale: 4 465 476
Accessi/giorno: (4 465 476) * 978/giorno = 4 367 235 528

U9 – Crea Scheda

- Personal Trainer 50 (L)
- Assegnare 950 (L)
- Clienti: 950 (L)
- Attiva: 950 (W)
- Scheda Attiva: 950 (W)
- Associare 950 (W)

Costo Totale: 7 650
Accessi/giorno: (7 650) * 2/giorno = 15 300

U10 – Archivia Scheda

- Clienti: 950 (L)
- Attiva: 713 (L)
- Scheda Attiva Completata: 713 (L)
- Scheda Archiviata: 16 150 (W)
- Associare: 950 (W)
- Attiva: 950 (W)

Costo Totale: 38 476
Accessi/giorno: (38 476) * 2/giorno = 76 952

U11 – Genera Report

- Personal Trainer: 50 (L)
- Assegnare: 950 (L)
- Clienti: 950 (L)
- Associare: 17 100 (L)
- Scheda: 17 100 (L)
- Eseguita: 547 200 (L)
- Sessione di Allenamento: 547 200 (L)
- Esercizi Svolti 3 283 200 (L)
- Scheda Esercizi 85 500 (L)

Costo Totale: 4 499 250
Accessi/giorno: (4 499 250) * 1/giorno = 4 499 250

U12 – Inserisci Esercizio
- Personal Trainer 50 (L)
- Assegnare 950 (L)
- Clienti: 950 (L)
- Attiva: 950 (L)
- Scheda Attiva: 950 (L)
- Associare 950 (L)
- Scheda Esercizi 85 500(W)

Costo Totale: 90 300
Accessi/giorno: (90 300) * 10/giorno = 903 000

U13 – Visualizza Clienti
- PT: 50 (L)
- Associare 950 (L)
- Clienti: 950 (L)

Costo Totale: 1 950
Accessi/giorno: (1 950) * 2/giorno = 3 900

AM1 – Registra Cliente e Associa con Personal Trainer
- Cliente: 950 (W)
- Assegnare: 950 (W)
- Personal Trainer: 50 (L)

Costo Totale: 3850
Accessi/giorno: (3850) * 0.56/giorno = 2 177

AM2 – Registrazione PT
- PT: 50 (W)

Costo Totale: 100
Accessi/giorno: (100) * 0.006/giorno ≃ 0.6

AM3 – Visualizza Esercizi
- Esercizi: 50 (L)

Costo Totale: 50
Accessi/giorno: (50) * 0.6/giorno ≃ 30

AM4 – Aggiungi Esercizio
- Esercizi: 50 (W)

Costo Totale: 100
Accessi/giorno: (100) * 0.006/giorno ≃ 0.6/giorno

L1 – Login
- Utente: 950 (L)
- PT: 50 (L)

Costo Totale: 1000
Accessi/giorno: (1 000) * 10/giorno = 10 000

Ristrutturazione dello schema E-R

Eliminazione delle generalizzazioni

Partendo dalla generalizzazione riguardante le schede attive completate e non completate, possiamo aggiungere un attributo chiamato "Tipo" alle schede attive. Questo attributo specifica se una scheda attiva è "completata" o "non completata". Inoltre, collegheremo la relazione "Attiva" al padre per stabilire la relazione gerarchica tra le due entità. In questo modo, avremo una struttura più chiara e organizzata per gestire le diverse tipologie di schede attive.

Adesso consideriamo adesso l'entità Scheda analizziamo, dunque, la scelta migliore per eliminare la generalizzazione fra schede attive e archiviate:

- La scelta più naturale risulterebbe portare il padre nei figli, questo principalmente perché l'entità Scheda ha due figli che supportano operazioni diverse e che quindi tendono a sottolineare la differenza fra le due entità.
  Il grande problema di questa opzione però riguarda l'eccessivo costo che andrei a pagare in memoria, dato che si duplicherebbero quasi tutte le associazioni, che, essendo legate al padre, passerebbero ad entrambi i figli.

- L'utilizzo di associazioni fra il padre e le figlie comporta, anch'esso, un grosso spreco di memoria dato che dovendo tenere traccia di queste due relazioni, gli accessi per le operazioni U9, U10 risulterebbero notevolmente maggiori. Esempio, per l'operazione U10 (archiviazione di una scheda):

Concetto	Costrutto	Accesso	Tipo
Clienti	E	950	L
Attiva	R	713	L
Scheda Attiva	E	713	L
Scheda-SchedaAttiva	R	713	L
Scheda-SchedaArchiviata	R	16 150	W
Scheda Archiviata	E	16 150	W
Scheda-SchedaAttiva	R	16 150	W
Attiva	R	713	W

Associare	R	16 150	L
Associare	R	16 150	W

Costo Totale: 149 628

Accessi/giorno: (149 628) * 2/giorno = 299 256

- Se invece utilizzassimo l'accorpamento dei figli nel genitore si verifica un piccolo spreco di memoria, dato che le operazioni che richiedono la Scheda Attiva dovranno prima calcolarla, ma ciò non costituisce un grande problema. Inoltre così facendo, diminuirebbe il numero di accessi, così come la duplicazione dei dati, migliorando l'efficienza dello spazio di archiviazione e semplificando la gestione e la manutenzione dei dati.

  Di conseguenza adottiamo la seguente strategia:

  1) Aggiungere l'attributo Data fine a Scheda con cardinalità (0,1) per indicare se una Scheda è Attiva o Archiviata.
  2) Dovrebbe essere aggiunta una relazione Attiva tra Scheda e Clienti per identificare la Scheda Attiva, ma ciò comporterebbe la presenza di un'associazione ciclica e, tra l'altro, ridondante. Infatti, è possibile ricavare la Scheda Attiva attraverso la data di inizio delle Schede. La Scheda con data di inizio maggiore è la Scheda Attiva. Bisogna aggiungere due vincoli:
     - La Scheda Attiva non deve avere una data di fine.
     - Ogni Cliente deve avere al massimo una Scheda Attiva.
  3) Aggiungere un vincolo per cui ogni Scheda Archiviata deve avere una data di fine.

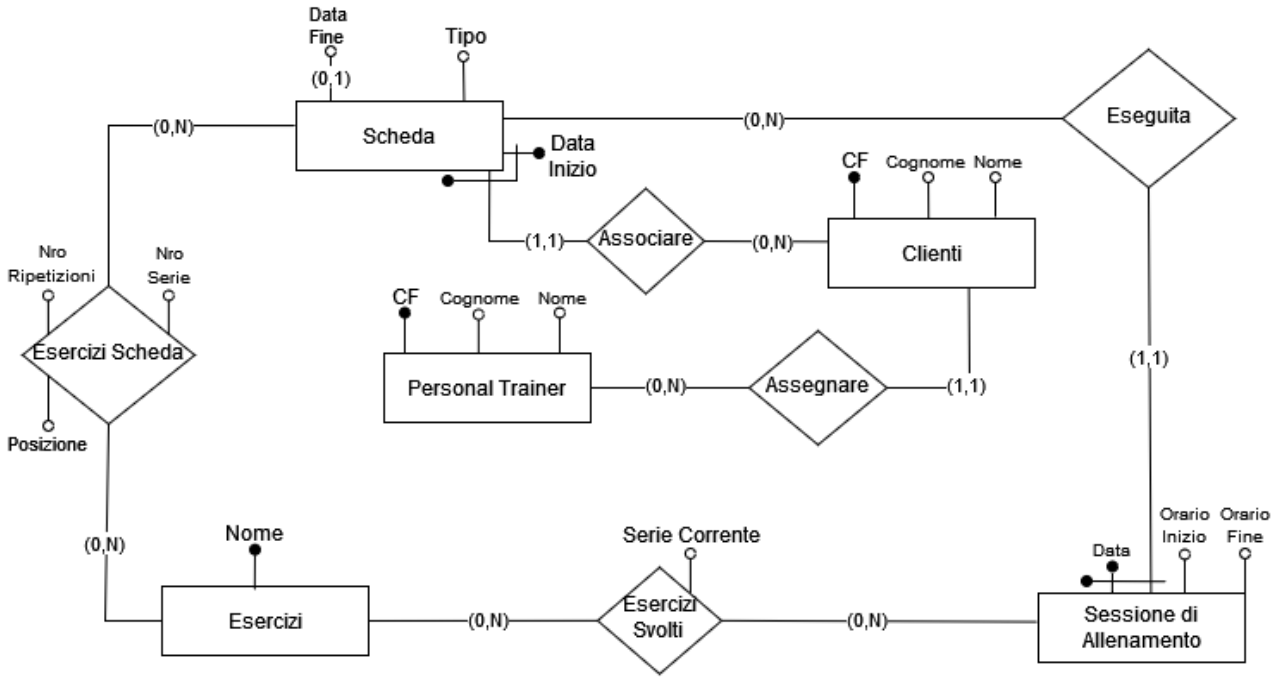
  L'operazione U10 diverrebbe:

Concetto	Costrutto	Accesso	Tipo
Clienti	E	950	L
Associare	R	17 100	L
Associare	R	16 150	W
Scheda	E	17 100	L
Scheda	E	16 150	W

Costo Totale: 67 450

Accessi/giorno: (67 450) * 2/giorno = 134 900

Schema Ristrutturato



Traduzione di entità e associazioni

Partendo dallo schema ristrutturato di cui sopra, costruiamo le seguenti relazioni per il modello relazionale

- *PersonalTrainer* (CF, Nome, Cognome)

- *Cliente* (CF, Nome, Cognome, PersonalTrainer)
  - *Cliente* (PersonalTrainer) ⊆ *PersonalTrainer* (CF)

- *Scheda* (DataInizio, Cliente, Tipo, DataFine*)
  - *Scheda* (Cliente) ⊆ *Cliente* (CF)

- *SessioneDiAllenamento* (DataSessione, Cliente, dataInizioScheda, OrarioInizio, OrarioFine*)
  - *SessioneDiAllenamento* (Cliente, DataInizioScheda) ⊆ *Scheda* (Cliente, DataInizio)

● *Esercizi* (Nome)

● *EserciziSvolti* (DataSessione, Cliente, dataInizioScheda, Esercizio, SerieCorrente)

○ *EserciziSvolti*(Cliente, dataInizioScheda, DataSessione) ⊆ *SessioneDiAllenamento*(Cliente, dataInizioScheda, DataSessione)

○ *EserciziSvolti* (Esercizio) ⊆ *Esercizi* (Nome)

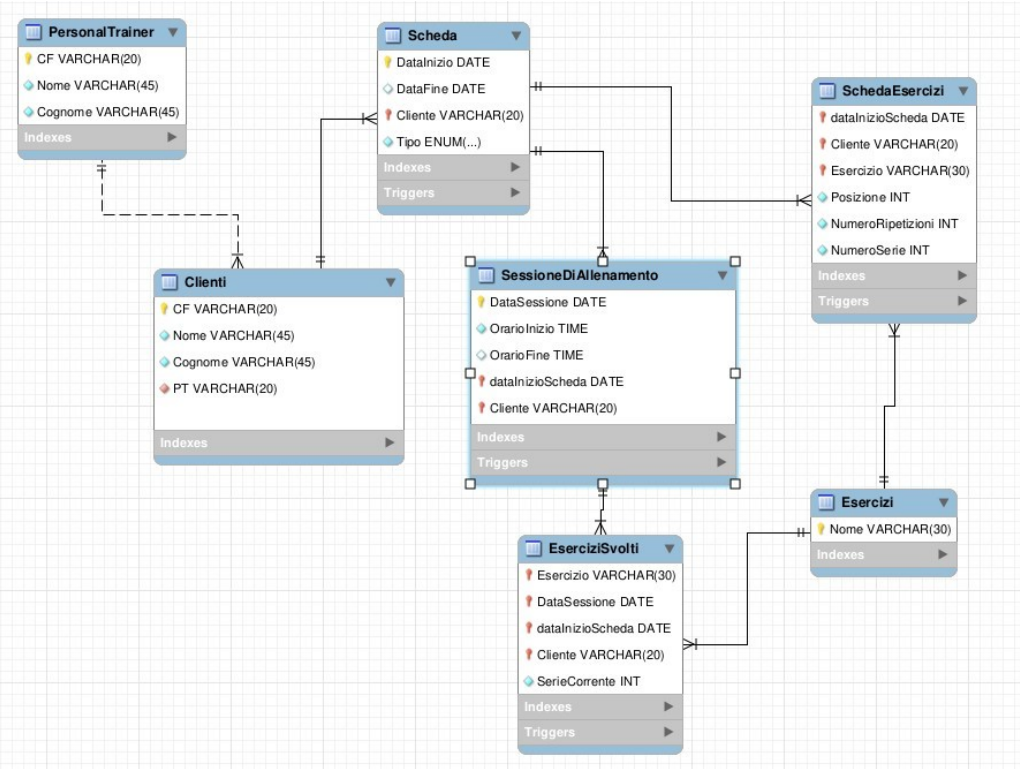
● *SchedaEsercizi* (Cliente, DataInizioScheda, Esercizio, NumeroRipetizioni, NumeroSerie, Posizione)

○ *EserciziScheda*(Cliente, DataInizioScheda) ⊆ *SessioneDiAllenamento*(Cliente, DataInizioScheda)

○ *EserciziScheda*(Esercizio) ⊆ *Esercizi*(Nome)

---

* Tutti i valori contrassegnati dall'asterisco possono accettare valori nulli



Normalizzazione del modello relazionale

1NF

Ricordando che una relazione soddisfa la 1NF se
- e presente chiave primaria
- non ci sono attributi ripetuti
- gli attributi non sono strutture complesse

Le tabelle fornite soddisfano la prima forma normale, in quanto ogni colonna contiene un singolo valore, non ci sono ripetizioni o gruppi di colonne e le chiavi esterne collegano correttamente le tabelle.

2NF

Ricordando che una relazione e in 2NF se la chiave scelta non contiene chiavi più piccole (ovvero la chiave scelta e effettivamente una chiave e non una superchiave).
In tutte le relazioni le chiavi non contengono chiavi più piccole e quindi sono minimali.

3NF

Una relazione e in terza forma normale se per ogni dipendenza funzionale X → A vale una delle seguenti:

• X e superchiave della relazione

• A appartiene ad almeno una chiave

Nel nostro caso questo vale per tutte le relazioni.

5. Progettazione fisica

Utenti e privilegi

Sono previsti 4 ruoli:

- Login
  - Grant in esecuzione su L1.
- Clienti
  - Grant in esecuzione su operazioni U1-U8.
- Personal Trainer
  - Grant in esecuzione su operazioni U9-U13.
- Segreteria
  - Grant in esecuzione su operazioni AM1 - AM4.

Inserimento/Modifica tabelle

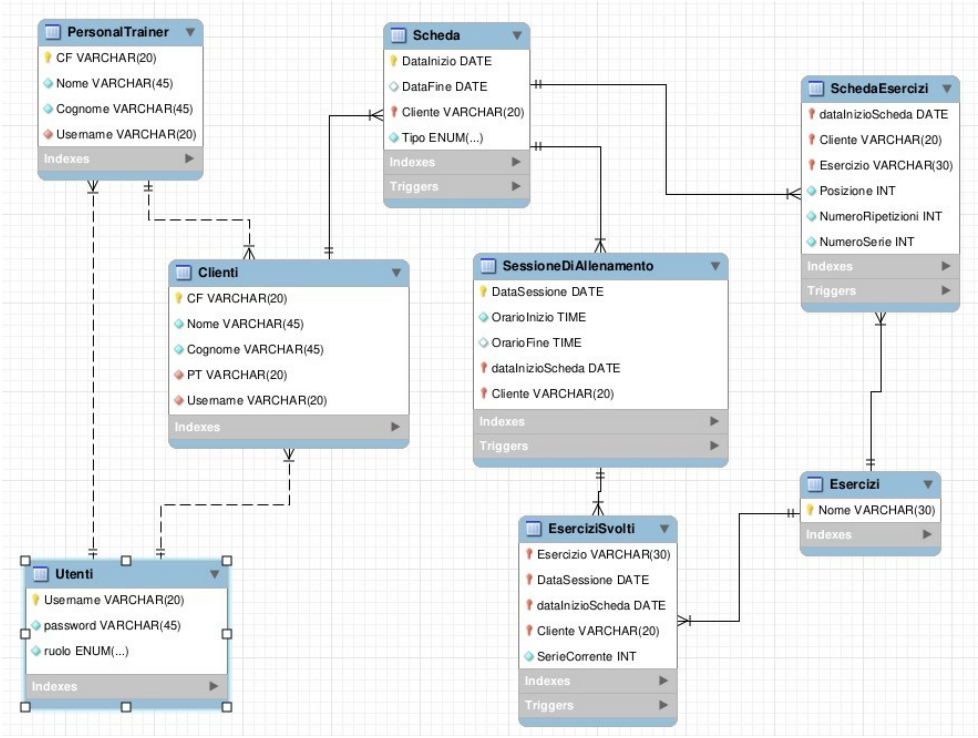
• Aggiungo una tabella chiamata “Utenti” che conterrà le seguenti colonne:

1) “Username” (campo per l’username dell’utente).

2) “Password” (campo per la password dell’utente).

3) “Ruolo” (campo per il ruolo dell'utente, rappresentato come un’enumerazione) Questi campi permetteranno l’accesso al sistema come utenti del servizio.

• Aggiungo un campo Username alle tabelle Clienti e Personal Trainer.



Strutture di memorizzazione

Tabella <Utenti>		
Attributo	**Tipo di dato**	**Attributi[2]**
Username	VARCHAR (20)	PK, NN
Password	VARCHAR (45)	NN
Ruolo	ENUM (‘pt’, ‘segreteria’, ‘clienti’)	NN

Tabella <Clienti>		
Colonna	**Tipo di dato**	**Attributi**
CF	VARCHAR (20)	PK
Nome	VARCHAR (45)	NN
Cognome	VARCHAR (45)	NN
PersonalTrainer	VARCHAR (20)	NN
Username	VARCHAR (20)	UQ, NN

Tabella <PersonalTrainer>		
Colonna	**Tipo di dato**	**Attributi**
CF	VARCHAR (20)	PK, NN
Nome	VARCHAR (45)	NN
Cognome	VARCHAR (45)	NN
Username	VARCHAR (20)	UQ, NN

[2] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

Tabella <Scheda>		
Colonna	**Tipo di dato**	**Attributi**
DataInizio	DATE	PK, NN
Cliente	VARCHAR (20)	PK, NN
DataFine	DATE	
Ruolo	ENUM (‘completata’, ‘non completata’)	NN

Tabella <Esercizi>		
Colonna	**Tipo di dato**	**Attributi**
Nome	VARCHAR (30)	PK, NN

Tabella <EserciziScheda>		
Colonna	**Tipo di dato**	**Attributi**
Cliente	VARCHAR (20)	PK, NN
DataInizioScheda	DATE	PK, NN
Esercizio	VARCHAR (30)	PK, NN
NumeroRipetizioni	INT	NN
NumeroSerie	INT	NN
Posizione	INT	NN

Tabella <SessioneDiAllenamento>		
Colonna	**Tipo di dato**	**Attributi**
Data	DATE	PK, NN
Cliente	VARCHAR (20)	PK, NN
DataInizioScheda	DATE	PK, NN
OrarioInizio	TIME	NN
OrarioFine	TIME	NN

Tabella <EserciziSvolti>		
Colonna	**Tipo di dato**	**Attributi**
DataSessione	DATE	PK, NN
Cliente	VARCHAR (20)	PK, NN
DataInizioScheda	DATE	PK, NN
Esercizio	VARCHAR (30)	PK, NN
SerieCorrente	INT	NN

Indici

Si omettono gli indici di tipo PRIMARY per le chiavi e INDEX per le foreign key, in quanto generati automaticamente.

Trigger

check_eserciziSvolti_insert

Il trigger è stato creato per impedire che un cliente esegua un esercizio che non è presente nella sua Scheda Attiva.

```
CREATE DEFINER=`root`@`localhost` TRIGGER `check_eserciziSvolti_insert` BEFORE INSERT ON `EserciziSvolti` FOR EACH ROW BEGIN
     DECLARE esercizio_count INT;


    -- Controlla se l'esercizio esiste nella tabella SchedaEsercizi
    SELECT COUNT(*) INTO esercizio_count
    FROM SchedaEsercizi
    WHERE Cliente = NEW.Cliente
        AND dataInizioScheda = NEW.dataInizioScheda
        AND Esercizio = NEW.Esercizio;

    -- Se l'esercizio non esiste, genera un errore
    IF esercizio_count = 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'L\'esercizio non esiste nella scheda.';
    END IF;
END
```

check_eserciziSvolti_orarioInsert

In accordo con la palestra, non si possono effettuare operazioni che riguardano “aggiornamenti” o “inserimenti” di esercizi nelle sessioni di allenamento dei clienti fra le 23:00 e le 7:00 del giorno successivo (orario di chiusura della palestra).

```
CREATE DEFINER=`root`@`localhost` TRIGGER `check_eserciziSvolti_orarioInsert` BEFORE INSERT ON `EserciziSvolti` FOR EACH ROW
BEGIN
    IF HOUR(CURRENT_TIME()) < 7 OR HOUR(CURRENT_TIME()) > 23 THEN
        SIGNAL SQLSTATE '45001'
        SET MESSAGE_TEXT = 'L\'orario di inserimento deve essere compreso tra le 7:00 e le 23:00';
    END IF;
END;
```

before_update_esercizisvolti

Il trigger è stato creato per evitare che un cliente esegua un numero di serie eccessivo di un esercizio rispetto a quanto specificato nella sua Scheda Attiva.

```
CREATE DEFINER=`root`@`localhost` TRIGGER `before_update_esercizisvolti` BEFORE UPDATE ON `EserciziSvolti` FOR EACH ROW BEGIN
```

```
    DECLARE num_serie INT;

    -- Ottieni il numero di serie dall'esercizischeda
    SELECT NumeroSerie INTO num_serie
    FROM SchedaEsercizi
    WHERE Esercizio = NEW.Esercizio
    AND dataInizioScheda = NEW.dataInizioScheda
    AND Cliente = NEW.Cliente;

    -- Verifica se la serie corrente è maggiore del numero di serie
    IF NEW.SerieCorrente > num_serie THEN
        SIGNAL SQLSTATE '45002'
            SET MESSAGE_TEXT = 'La serie corrente è maggiore del numero di serie consentito.';
    END IF;
END
```

check_eserciziSvolti_orarioUpdate

In accordo con la palestra, non si possono effettuare operazioni che riguardano “aggiornamenti” o “inserimenti” di esercizi nelle sessioni di allenamento dei clienti fra le 23:00 e le 7:00 del giorno successivo (orario di chiusura della palestra).

```
CREATE DEFINER=`root`@`localhost` TRIGGER `check_eserciziSvolti_orarioUpdate` BEFORE UPDATE ON `EserciziSvolti` FOR EACH ROW
BEGIN
        IF HOUR(CURRENT_TIME()) < 7 OR HOUR(CURRENT_TIME()) > 23 THEN
        SIGNAL SQLSTATE '45003'
        SET MESSAGE_TEXT = 'L\'orario di aggiornamento deve essere compreso tra le 7:00 e le 23:00';
    END IF;
END;
```

Scheda_BEFORE_INSERT

Il trigger è stato creato per garantire che ogni cliente abbia una sola scheda attiva.

```
CREATE DEFINER=`root`@`localhost` TRIGGER `Scheda_BEFORE_INSERT` BEFORE INSERT ON `Scheda` FOR EACH ROW BEGIN
    -- Dichiarazione delle variabili
    DECLARE var_count INT;

    -- Verifica se esistono altre tuple per lo stesso cliente con DataFine NULL
    SELECT COUNT(*) INTO var_count
    FROM Scheda
    WHERE Cliente = NEW.Cliente AND DataFine IS NULL;

    -- Se esistono tuple con DataFine NULL, genera un errore
    IF var_count > 0 THEN
        SIGNAL SQLSTATE '45004' SET MESSAGE_TEXT = 'Esiste già una scheda attiva per questo cliente.';
    END IF;
```

```
END
```

before_update_scheda

Il trigger è stato creato per garantire che ogni scheda archivata, sia prima completata.

```
CREATE DEFINER=`root`@`localhost` TRIGGER `before_update_scheda` BEFORE UPDATE ON `Scheda` FOR EACH ROW BEGIN
    IF New.Tipo = 'non completata' THEN
            SIGNAL SQLSTATE '45005' SET MESSAGE_TEXT = 'La scheda deve essere completa prima di essere archiviata.';
    END IF;
END
```

SchedaEsercizi_BEFORE_INSERT

Il trigger è stato creato per garantire che ogni esercizio, in ogni scheda, abbia una posizione diversa.

```
CREATE DEFINER=`root`@`localhost` TRIGGER `SchedaEsercizi_BEFORE_INSERT` BEFORE INSERT ON `SchedaEsercizi` FOR EACH ROW BEGIN
   DECLARE position_count INT;

    -- Controlla se esiste un esercizio con la stessa posizione e dataInizioScheda
    SELECT COUNT(*) INTO position_count
    FROM SchedaEsercizi
    WHERE dataInizioScheda = NEW.dataInizioScheda
        AND Posizione = NEW.Posizione
        AND Cliente = NEW.Cliente;

    -- Se esiste un esercizio con la stessa posizione, genera un errore
    IF position_count > 0 THEN
        SIGNAL SQLSTATE '45006'
        SET MESSAGE_TEXT = 'Esiste già un esercizio con la stessa posizione in questa scheda.';
    END IF;
END
```

SessioneDiAllenamento_BEFORE_INSERT

In accordo con la palestra, non si possono effettuare operazioni che riguardano “aggiornamenti” o “inserimenti” di esercizi nelle sessioni di allenamento dei clienti fra le 23:00 e le 7:00 del giorno successivo (orario di chiusura della palestra).

```
CREATE DEFINER = CURRENT_USER TRIGGER `PROVADB`.`SessioneDiAllenamento_BEFORE_INSERT` BEFORE INSERT ON `SessioneDiAllenamento` FOR EACH ROW
BEGIN
        IF HOUR(CURRENT_TIME()) < 7 OR HOUR(CURRENT_TIME()) > 23 THEN
        SIGNAL SQLSTATE '45007'
        SET MESSAGE_TEXT = 'Puoi iniziare una sessione solo tra le 7:00 e le 23:00';
    END IF;
END
```

SessioneDiAllenamento_BEFORE_UPDATE

In accordo con la palestra, non si possono effettuare operazioni che riguardano “aggiornamenti” o “inserimenti” di esercizi nelle sessioni di allenamento dei clienti fra le 23:00 e le 7:00 del giorno successivo (orario di chiusura della palestra).

```
CREATE DEFINER = CURRENT_USER TRIGGER `PROVADB`.`SessioneDiAllenamento_BEFORE_UPDATE` BEFORE UPDATE ON `SessioneDiAllenamento` FOR EACH ROW
BEGIN
        IF HOUR(CURRENT_TIME()) < 7 OR HOUR(CURRENT_TIME()) > 23 THEN
        SIGNAL SQLSTATE '45008'
        SET MESSAGE_TEXT = 'Puoi chiudere una sessione solo tra le 7:00 e le 23:00';
    END IF;
END
```

Eventi

clean_routines

È stato aggiunto un evento, chiamato clean_routines, al database per gestire il problema dell’eccessiva quantità di dati e prevenire problemi di prestazioni. Questo evento si occupa di eliminare automaticamente tutte le schede che risalgono a più di 5 anni fa. Poiché le chiavi esterne sono configurate con l’opzione ON DELETE CASCADE, questa operazione comporterà anche l’eliminazione automatica di tutte le sessioni di allenamento associate alle schede, degli esercizi collegati a tali schede e di tutti gli esercizi svolti durante le sessioni di allenamento. In questo modo, viene mantenuta la coerenza del database e si previene l'accumulo di dati obsoleti.

```
CREATE EVENT `clean_routines`
ON SCHEDULE
    EVERY 1 YEAR
    STARTS CURRENT_TIMESTAMP + INTERVAL 5 YEAR
    ON COMPLETION PRESERVE
COMMENT 'Remove routines older than 5 years'
DO
    DELETE FROM `Scheda` WHERE `DataInizio` < (NOW() - INTERVAL 5 YEAR);
```

clean_notFinishedTrainingSessions

È stato aggiunto un evento chiamato “clean_notFinishedTrainingSessions” al database per gestire il problema delle sessioni di allenamento non chiuse durante l’orario di apertura della palestra. Questo evento viene eseguito ogni giorno alle 23:30, quando la palestra è chiusa e nessun’altra sessione di allenamento può essere eseguita. Scopo dell’evento è eliminare tutte le sessioni di allenamento che non hanno un orario di fine, al fine di evitare di occupare memoria inutilmente.

```
CREATE EVENT clean_notFinishedTrainingSessions
ON SCHEDULE
        EVERY 1 DAY
        STARTS CURRENT_DATE + INTERVAL 1 DAY + INTERVAL '23:30' HOUR_MINUTE
        COMMENT 'Remove not finished training sessions every day at 23:30'
DO
        DELETE FROM SessioniDiAllenamento WHERE OrarioFine IS NULL
```

Viste

Non sono state utilizzate viste.

Funzioni

check_cf

Funzione utile a riconoscere Codici Fiscali Validi

```
CREATE DEFINER=`root`@`localhost` FUNCTION `check_cf`(CF VARCHAR(20)) RETURNS tinyint(1)
    DETERMINISTIC
BEGIN


    IF CF REGEXP '^[A-Z]{6}[0-9]{2}[A-Z][0-9]{2}[A-Z][0-9]{3}[A-Z]$' THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
END
```

check_init_session

Funzione utile a identificare già sessioni iniziate

```
CREATE DEFINER=`root`@`localhost` FUNCTION `check_init_session`(CF VARCHAR(20), dataInitScheda date) RETURNS
int
    DETERMINISTIC
BEGIN
        declare var_result INT;

        IF EXISTS (
                SELECT 1
                FROM SessioneDiAllenamento
                WHERE Cliente = CF
                        AND dataInizioScheda = dataInitScheda
                        AND OrarioInizio IS NOT NULL
                        AND OrarioFine IS NULL
        ) THEN
                SET var_result = 1; -- 1 = Sessione Iniziata ma non finita
        ELSEIF EXISTS (
                SELECT 1
```

```
                FROM SessioneDiAllenamento
                WHERE Cliente = CF
                        AND dataInizioScheda = dataInitScheda
                        AND OrarioInizio IS NOT NULL
                        AND OrarioFine IS NOT NULL
        ) THEN
                SET var_result = 2; -- 2 = Sessione Iniziata e Finita
        ELSE
                SET var_result = 3; -- 3 = Sessione non iniziata
        END IF;
        RETURN var_result;
END
```

Stored Procedures e transazioni

aggiungi_cliente

Procedura introdotta per aggiungere Clienti al Database; è stato deciso di inserire una transazione per far sì che i due inserimenti di utenti e clienti avvengano “o entrambi o nessuno dei due”.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `aggiungi_cliente`(
        in var_codiceFiscale VARCHAR(20),
        in var_nome VARCHAR(45),
        in var_cognome VARCHAR(45),
        in var_username VARCHAR(20),
        in var_pass VARCHAR (45))
BEGIN

        declare var_PT VARCHAR(20);
        declare exit handler for sqlexception
        begin
                rollback; ## Annullamento Transazione
                resignal; ## Ridirezione Segnale al Client
        end;

        IF (PROVADB.check_cf(var_codiceFiscale) is FALSE) THEN
                signal sqlstate '45009' set message_text = 'CODICE FISCALE NON VALIDO';
        END IF;
        SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
        START TRANSACTION ;

                -- Selezione PT con meno clienti
                SELECT PersonalTrainer.CF into var_PT
                FROM PersonalTrainer
                LEFT JOIN Clienti ON PersonalTrainer.CF = Clienti.PT
                GROUP BY PersonalTrainer.CF
                ORDER BY COUNT(Clienti.PT) ASC
                LIMIT 1;
```

```
                insert into `Utenti` (`Username`, `password`, `ruolo`)
                values (var_username, sha1(var_pass), 'clienti');

                -- Aggiungo Cliente
                insert into `Clienti` (`CF`, `Nome`, `Cognome`, `Username`, `PT`)
                values (var_codiceFiscale, var_nome, var_cognome, var_username, var_PT);

        COMMIT;
END
```

aggiungi_esercizio_in_palestra

Procedura introdotta per aggiungere Esercizi al Database.

```
CREATE PROCEDURE `aggiungi_esercizio_in_palestra` (in var_esercizio varchar(30))
BEGIN
        insert into `Esercizi` (`Nome`)
        values (var_esercizio);
END
```

aggiungi_pt

Procedura introdotta per aggiungere Personal Trainer al Database; è stato deciso di inserire una transazione per far sì che i due inserimenti di utenti e Personal Trainer avvengano “o entrambi o nessuno dei due”.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `aggiungi_pt`(
        in var_codiceFiscale VARCHAR(20),
        in var_nome VARCHAR(45),
        in var_cognome VARCHAR(45),
        in var_username VARCHAR(20),
        in var_pass VARCHAR (45))
BEGIN

        IF (PROVADB.check_cf(var_codiceFiscale) is FALSE) THEN
                signal sqlstate '45010' set message_text = 'CODICE FISCALE PT NON VALIDO';
        END IF;

        SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
        START TRANSACTION ;
                insert into `Utenti` (`Username`, `password`, `ruolo`)
                values (var_username, sha1(var_pass), 'pt');

                insert into `PersonalTrainer` (`CF`, `Nome`, `Cognome`, `Username`)
```

```
                values (var_codiceFiscale, var_nome, var_cognome, var_username);
        COMMIT;
END
```

aggiungi_serie_esercizio

Procedura introdotta per aggiornare il nuovo numero di SerieCorrenti in EserciziSvolti

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `aggiungi_serie_esercizio`(in var_nomeEsercizio varchar(30),
        in var_dataSessione date,
        in var_dataInizioScheda date,
        in var_cfClienti varchar(20),
        in var_serieCorrente int)
BEGIN

    declare var_serieAggiornata int;
    declare exit handler for sqlexception
    begin
        rollback; ## Annullamento Transazione
        resignal; ## Ridirezione Segnale al Client
    end;
    SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
    START TRANSACTION ;

        set var_serieAggiornata = var_seriecorrente + 1;


        UPDATE EserciziSvolti
        SET  SerieCorrente = var_serieAggiornata
        WHERE Cliente = var_cfClienti
                AND  dataInizioScheda = var_dataInizioScheda
                AND Esercizio = var_nomeEsercizio
                AND dataSessione = var_dataSessione;
    COMMIT;
END
```

archivia_scheda

Procedura introdotta per permettere ad un Personal Trainer di archiviare una Scheda Attiva Completa

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `archivia_scheda`(
    IN var_cfCliente VARCHAR(20),
    IN var_cfPt VARCHAR(20)
)
BEGIN
    DECLARE var_dataFine DATE;
    DECLARE var_dataInizio DATE;
    DECLARE var_sessioneStato INT;
    DECLARE var_Pt VARCHAR(20);


    DECLARE EXIT HANDLER FOR SQLEXCEPTION
```

```
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;

    -- Imposta il livello di isolamento appropriato per la transazione
    SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
    START TRANSACTION;

                 IF (PROVADB.check_cf(var_cfCliente) is FALSE) THEN
                        signal sqlstate '45011' set message_text = 'CODICE FISCALE NON VALIDO';
                END IF;

                SELECT PT INTO var_Pt
                FROM Clienti
                WHERE CF = var_cfCliente;

        IF (var_Pt <> var_cfPt) THEN
                        signal sqlstate '45012' set message_text = 'NON PUOI ARCHIVIARE LA SUA SCHEDA, NON E\'
UN TUO CLIENTE.';
                END IF;

        -- Ottenere la data di inizio della scheda attiva per il cliente
                SELECT DataInizio INTO var_dataInizio
                FROM Scheda join Clienti on Cliente = CF
                WHERE Cliente = var_cfCliente AND DataFine IS NULL;

        IF (var_dataInizio is NULL) THEN
                        signal sqlstate '45013' set message_text = 'Il cliente selezionato non ha una scheda
attiva.';
                END IF;

        -- La sessione di allenamento non è iniziata, si può procedere con l'archiviazione della scheda
        SET var_dataFine = CURDATE();

                IF (PROVADB.check_init_session(var_cfCliente, var_dataInizio) = 1) THEN
                        signal sqlstate '450014' set message_text = 'ATTENZIONE L\'UTENTE IN QUESTIONE STA
ESEGUENDO UNA SESSIONE CON LA SCHEDA CHE VUOI ARCHIVIARE: ASPETTA CHE FINISCA.';
                END IF;

        UPDATE Scheda
        SET DataFine = var_dataFine
        WHERE Cliente = var_cfCliente AND DataFine IS NULL;

        COMMIT;
END
```

crea_nuova_scheda

Procedura introdotta per permettere ad un Personal Trainer di creare una Scheda Attiva

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `crea_nuova_scheda`(
        in var_cfCliente VARCHAR(20),
        in var_cfPt VARCHAR(20),
    out var_dataInizio date)
BEGIN
    declare var_Pt VARCHAR(20);


        DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;


    SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
    START TRANSACTION;
                IF (PROVADB.check_cf(var_cfCliente) is FALSE) THEN
                        signal sqlstate '45015' set message_text = 'CODICE FISCALE NON VALIDO';
                END IF;


                SELECT PT INTO var_Pt
                FROM Clienti
                WHERE CF = var_cfCliente;


        IF (var_Pt <> var_cfPt) THEN
                        signal sqlstate '45016' set message_text = 'NON PUOI CREARGLI LA SCHEDA, NON E\' UN
TUO CLIENTE.';
                END IF;


                set var_dataInizio = curdate();


                INSERT INTO `Scheda` (`DataInizio`, `Cliente`, `Tipo`)
                values (var_dataInizio, var_cfCliente, 'non completata');


    COMMIT;
END
```

esegui_esercizio

Procedura introdotta per permettere ad un cliente di eseguire un esercizio in una sessione di allenamento

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `esegui_esercizio`(in var_nomeEsercizio varchar(30),
        in var_dataSessione date,
        in var_dataInizioScheda date,
        in var_cfClienti varchar(20))
```

```
BEGIN
        declare exit handler for sqlexception
    begin
        rollback; ## Annullamento Transazione
        resignal; ## Ridirezione Segnale al Client
    end;

        insert   into   `EserciziSvolti`   (`Esercizio`,   `DataSessione`,   `dataInizioScheda`,   `Cliente`,
`SerieCorrente`)
        values (var_nomeEsercizio, var_dataSessione, var_dataInizioScheda, var_cfClienti, 1);

END
```

genera_report

La seguente stored procedure è una delle più complesse. Essa consente a un Personal Trainer di generare un report dettagliato su tutti i suoi clienti, includendo tutte le sessioni di allenamento svolte da ciascun cliente. Il report fornisce informazioni come la durata della sessione e le relative percentuali di completamento della scheda assegnata, nel periodo di tempo specificato dal Personal Trainer.

Durante l'esecuzione della stored procedure, viene creata una tabella temporanea per memorizzare le informazioni relative alle sessioni svolte durante il periodo specificato nei parametri iniziali. Di seguito sono elencati i dettagli inclusi nella tabella temporanea:

- Codice Fiscale del cliente
- Nome del cliente
- Cognome del cliente
- Data di inizio della scheda assegnata al cliente
- Data in cui è stata svolta la sessione
- Percentuale di completamento della scheda
- Durata della sessione

Queste informazioni consentono al Personal Trainer di ottenere una panoramica completa delle attività svolte dai propri clienti durante il periodo selezionato.

Va sottolineato anche il livello di isolamento raggiunto per questa transazione, che è di tipo “Read Uncommitted”. Questo livello di isolamento è stato scelto perché tutte le letture effettuate nel report riguardano dati che nessuno può modificare contemporaneamente. Ciò significa che non c'è rischio

di incorrere in letture sporche o ripetibili, in quanto i dati letti non subiscono modifiche durante l'esecuzione del report.

È importante notare che non vengono considerate tutte le sessioni di allenamento “aperte”, in quanto queste avranno una percentuale di completamento ancora non definita e non sarebbero utili per il report. Pertanto, solo le sessioni di allenamento completate vengono prese in considerazione per ottenere una valutazione accurata delle prestazioni dei clienti nel periodo specificato.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `genera_report`(IN var_trainer_cf VARCHAR(20), IN var_dataInizio
DATE, IN var_dataFine DATE)
BEGIN

	declare exit handler for sqlexception ## Dichiarazione Gestore eccezione sollevata dal Trigger
    begin
        rollback; ## Annullamento Transazione
        resignal; ## Ridirezione Segnale al Client
    end;


        DROP TABLE IF EXISTS report_temp;


        -- Crea una tabella temporanea per il report
        CREATE TEMPORARY TABLE report_temp (
        CF VARCHAR(20),
        Nome VARCHAR(45),
        Cognome VARCHAR(45),
        DataScheda DATE,
        Sessione DATE,
        Completamento VARCHAR(8),
        Durata CHAR(10));



        SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
        START TRANSACTION;


                -- Inserisci i dati nel report temporaneo
                INSERT INTO report_temp (CF, Nome, Cognome, DataScheda, Sessione, Completamento, Durata)
                SELECT c.CF as CF, c.Nome as Nome, c.Cognome as Cognome, sa.DataInizio as DataScheda,
ses.DataSessione as Sessione,
                        CONCAT(IFNULL((SUM(esv.SerieCorrente) / SUM(es.NumeroSerie)) * 100, 0), '%') AS
Completamento,
                        CAST(TIMEDIFF(ses.OrarioFine, ses.OrarioInizio) AS CHAR(10)) AS Durata
                FROM Clienti c
                        JOIN Scheda sa ON c.CF = sa.Cliente
                        JOIN SessioneDiAllenamento ses ON sa.Cliente = ses.Cliente
                                AND sa.DataInizio = ses.dataInizioScheda
```

```
                        JOIN SchedaEsercizi es ON sa.Cliente = es.Cliente
                                AND sa.DataInizio = es.dataInizioScheda
                        LEFT JOIN EserciziSvolti esv ON ses.Cliente = esv.Cliente
                                AND ses.dataInizioScheda = esv.dataInizioScheda
                                AND ses.DataSessione = esv.dataSessione
                                AND es.Esercizio = esv.Esercizio
                WHERE c.PT = var_trainer_cf and OrarioFine is not NULL
                        AND ses.DataSessione BETWEEN var_dataInizio AND var_dataFine
                GROUP BY c.CF, c.Nome, c.Cognome, ses.DataSessione, sa.DataInizio;

                -- Seleziona i dati dal report temporaneo per visualizzarli
                SELECT *
                FROM report_temp;

    COMMIT;

END
```

inizia_sessione

Procedura introdotta per permettere ad un cliente di iniziare una sessione di allenamento.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `inizia_sessione`(in var_cfClienti varchar(20),
        out var_dataInizioScheda date,
        out var_oggi date,
        out var_ora time)
BEGIN

    declare var_tipo varchar(15);

    declare exit handler for sqlexception
        begin
                rollback;
                resignal;
        end;
    SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
    START TRANSACTION;

                SELECT DataInizio, Tipo INTO var_dataInizioScheda, var_tipo
                FROM Scheda
                WHERE DataFine IS NULL AND Cliente = var_cfClienti;

                if(var_tipo = 'non completata') then
                        signal sqlstate '45029' set message_text = 'NON PUOI INIZIARE UNA SESSIONE: LA SCHEDA
NON E\' ANCORA PRONTA.';
                END IF;
```

```
                set var_oggi = curdate();
                set var_ora = curtime();

                insert into `SessioneDiAllenamento` (`dataSessione`, `OrarioInizio`, `OrarioFine`,
`dataInizioScheda`, `Cliente`)
                values (var_oggi, var_ora, NULL, var_dataInizioScheda, var_cfClienti);
        commit;
END
```

inserisci_esercizio

Procedura introdotta per permettere ad un personal Trainer di inserire un singolo esercizio dentro una scheda attiva non completata.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `inserisci_esercizio`(
    IN var_cliente VARCHAR(20),
    IN var_esercizio VARCHAR(30),
    IN var_numeroRipetizioni INT,
    IN var_numeroSerie INT,
    IN var_posizione INT,
    IN var_dataInizioScheda DATE
)
BEGIN
        declare var int;
    declare exit handler for sqlexception
    begin
        rollback; ## Annullamento Transazione
        resignal; ## Ridirezione Segnale al Client
    end;
    SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
        start transaction;
                SELECT COUNT(*) INTO var
                FROM Scheda
                WHERE Cliente = var_cliente AND DataInizio = var_dataInizioScheda AND Tipo = 'completata' and
DataFine is NULL;

        IF (var = 1) THEN
                        signal sqlstate '45030' set message_text = 'NON PUOI INSERIRE ESERCIZI DENTRO SCHEDE
GIA\' COMPLETATE';
        ELSE
                        insert into `SchedaEsercizi` (`dataInizioScheda`, `Cliente`, `Esercizio`, `Posizione`,
`NumeroRipetizioni`, `NumeroSerie`)
                        values (var_dataInizioScheda, var_cliente, var_esercizio, var_posizione,
var_numeroRipetizioni, var_numeroSerie);
                END IF;
    commit;

END
```

login

Procedura di login. Viene selezionato il ruolo dell'utente di cui sono date le credenziali. Il ruolo viene poi mappato su un codice numerico tornato come OUT. Notare che viene confrontata la password presente nel sistema con la SHA1 della password data in input: quando viene inserito un utente, ad esempio un insegnante, la sua password non viene salvata in chiaro, ma ne viene salvato la SHA1.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `login`(
        IN var_username VARCHAR(15),
        IN var_password VARCHAR(20),
        OUT var_role INT)
BEGIN

        DECLARE var_enum_role ENUM("pt","segreteria","clienti") ;


        SELECT `Ruolo`
    FROM `Utenti`
    WHERE `Username` = var_username AND `Password` = SHA1(var_password)
    INTO var_enum_role ;


    IF var_enum_role = "pt" THEN
                SET var_role = 0 ;
        ELSEIF var_enum_role = "segreteria" THEN
                SET var_role = 1 ;
        ELSEIF var_enum_role = "clienti" THEN
                SET var_role = 2 ;
        ELSE
                SET var_role = 3 ;
        END IF ;


END
```

prendi_cliente_cf

Procedura che si appoggia al Login e permette di recuperare il Codice Fiscale di un Cliente.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `prendi_cliente_cf`(in var_username varchar(20), out var_cf varchar(20))
BEGIN
    SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
        START TRANSACTION ;
                SELECT CF into var_cf
                FROM Clienti
                WHERE Username = var_username;
        COMMIT;
END
```

prendi_cliente_cf

Procedura che si appoggia al Login e permette di recuperare il Codice Fiscale di un Personal Trainer.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `prendi_pt_cf`(in var_username varchar(20), out var_cf varchar(20))
BEGIN

	SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
	START TRANSACTION ;

		SELECT CF into var_cf
		FROM PersonalTrainer
		WHERE Username = var_username;

	COMMIT;

END
```

pt_visualizza_scheda_attiva

Procedura che permette ad un Personal Trainer di visualizzare la scheda attiva di un suo cliente.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `pt_visualizza_scheda_attiva`(in var_cfClienti varchar(20), in var_pt varchar(20))
BEGIN
	declare var_Dat date;
    declare var_num INT;
    declare exit handler for sqlexception
	begin
		rollback;
		resignal;
	end;
	SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
	start transaction;

		IF (PROVADB.check_cf(var_cfClienti) is FALSE) THEN
			signal sqlstate '45017' set message_text = 'CODICE FISCALE NON VALIDO';
		END IF;

		SELECT count(*) INTO var_num
		FROM Clienti
		WHERE PT = var_pt AND CF = var_cfClienti;

		IF (var_num = 0) THEN
			signal sqlstate '45018' set message_text = 'NON E\' UN TUO CLIENTE, MI SPIACE MA NON PUOI VEDERE LA SUA SCHEDA ATTIVA';
		END IF;

		SELECT DataInizio INTO var_Dat
		FROM Scheda
		WHERE DataFine IS NULL AND Cliente = var_cfClienti;
```

```
                IF var_Dat IS NULL THEN
                        SIGNAL SQLSTATE '45019' SET MESSAGE_TEXT = 'SCHEDA ATTIVA INESISTENTE';
                END IF;

        SELECT Posizione, Esercizio, NumeroSerie as Serie, NumeroRipetizioni as Ripetizioni
        FROM SchedaEsercizi
        WHERE Cliente = var_cfClienti and dataInizioScheda = var_Dat
        order by Posizione;


        commit ;
END
```

recupera_dati_sessione

Procedura che permette ad un Cliente di recuperare la sessione precedentemente iniziata e non finita.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `recupera_dati_sessione`(IN var_cliente VARCHAR(20),
        OUT var_dataSessione date,
        OUT var_dataInizioScheda date,
    OUT var_orarioInizio TIME)
BEGIN
    DECLARE num_sessioni INT;

    SELECT COUNT(*) INTO num_sessioni
    FROM SessioneDiAllenamento
    WHERE Cliente = var_cliente
        AND DataSessione = CURDATE()
        AND OrarioInizio IS NOT NULL
        AND OrarioFine IS NULL;

    IF num_sessioni > 0 THEN
                 SELECT DataInizio INTO var_dataInizioScheda
                FROM Scheda
                WHERE DataFine IS NULL AND Cliente = var_cliente;

                set var_dataSessione = curdate();


                SELECT OrarioInizio into var_orarioInizio
                FROM SessioneDiAllenamento
                WHERE Cliente = var_cliente
                        AND OrarioFine is null
                        AND dataInizioScheda = var_dataInizioScheda
                        AND DataSessione = var_dataSessione;

    ELSE
          signal sqlstate '45020' set message_text = 'IMPOSSIBILE RECUPERARE LA SESSIONE DI ALLENAMENTO: SI
POSSONO RECUPERARE SOLO SESSIONI INIZIATE MA NON FINITE';
    END IF;
```

```
END
```

scegli_scheda_archiviata

Procedura che permette ad un Cliente di scegliere quale scheda archiviata visualizzare.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `scegli_scheda_archiviata`(IN var_CF_Cliente VARCHAR(20), IN var_dataInizioScheda DATE)
BEGIN

    DECLARE numEsercizi INT;
        declare exit handler for sqlexception
        begin
                rollback;
                resignal;
        end;
        SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
    START TRANSACTION;
                SELECT COUNT(*) INTO numEsercizi
                FROM Scheda s
                JOIN SchedaEsercizi se ON s.DataInizio = se.DataInizioScheda AND s.Cliente = se.Cliente
                WHERE s.Cliente = var_CF_Cliente AND s.DataInizio = var_dataInizioScheda;

                IF numEsercizi = 0 THEN
                        signal sqlstate '45021' set message_text = 'DATA NON NEL DATABASE';
                ELSE
                        SELECT Posizione, Esercizio, NumeroSerie as Serie, NumeroRipetizioni as Ripetizioni
                        FROM Scheda s
                        JOIN SchedaEsercizi se ON s.DataInizio = se.DataInizioScheda AND s.Cliente = se.Cliente
                        WHERE s.Cliente = var_CF_Cliente AND s.DataInizio = var_dataInizioScheda
                        ORDER BY Posizione;
                END IF;
        COMMIT;
END
```

scegli_scheda_ attiva_non_completa

Procedura che permette ad un Personal Trainer di scegliere una scheda attiva, non completa. Questa procedura si appoggia successivamente all'inserimento di nuovi esercizi dato che è stata ideata per scegliere una scheda da poter ancora modificare, prima di farla eseguire da un cliente.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `scegli_scheda_attiva_non_completa`(IN var_CF_Cliente VARCHAR(20),
        out var_dataInizioScheda DATE,
    out var_Position int)
BEGIN

        DECLARE num INT;
```

```
        declare exit handler for sqlexception
        begin
                rollback;
                resignal;
        end;
        SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
        START TRANSACTION;
                IF (PROVADB.check_cf(var_CF_Cliente) is FALSE) THEN
                        signal sqlstate '45022' set message_text = 'CODICE FISCALE NON VALIDO';
                END IF;


                SELECT DataInizio INTO var_dataInizioScheda
                FROM Scheda
                WHERE Cliente = var_CF_Cliente AND DataFine is NULL;


                IF (var_dataInizioScheda IS NULL) THEN
                        signal sqlstate '45023' set message_text = 'IL CLIENTE SELEZIONATO NON HA PROPRIO UNA
SCHEDA ATTIVA';
                END IF;


                SELECT max(Posizione) into var_Position
                FROM SchedaEsercizi
                WHERE Cliente = var_CF_Cliente AND dataInizioScheda = var_dataInizioScheda;

                SELECT Posizione, Esercizio, NumeroSerie as Serie, NumeroRipetizioni as Ripetizioni
                FROM Scheda s
                JOIN SchedaEsercizi se ON s.DataInizio = se.DataInizioScheda AND s.Cliente = se.Cliente
                WHERE s.Cliente=var_CF_Cliente AND s.DataInizio=var_dataInizioScheda AND Tipo='non completata'
                ORDER BY Posizione;
        COMMIT;
END
```

scheda_completata

Procedura che permette ad un Personal Trainer di segnare una scheda attiva come completa.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `scheda_completata`(IN var_cfCliente VARCHAR(20), in
var_dataInizioScheda date)
BEGIN
        UPDATE Scheda
        SET Tipo = 'completata'
        WHERE DataFine is NULL AND Cliente = var_cfCliente;
END
```

termina_sessione

Procedura che permette ad un Cliente di terminare una sessione di allenamento.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `termina_sessione`(in var_dataSessione date,
        in var_OrarioInizio time,
    in var_dataInizioScheda date,
    in var_cfClienti varchar(20))
BEGIN

        declare var_time time;
        set var_time = curtime();

        UPDATE SessioneDiAllenamento
        SET  OrarioFine = var_time
        WHERE Cliente = var_cfClienti
                AND  dataInizioScheda = var_dataInizioScheda
                AND OrarioInizio = var_OrarioInizio
                AND dataSessione = var_dataSessione;

END
```

visualizza_clienti

Procedura che permette ad un Personal Trainer di visualizzare tutti i suoi clienti.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_clienti`(in var_cfPT VARCHAR(20))
BEGIN

        SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
        start transaction;

                SELECT CF, Nome, Cognome
                FROM Clienti
                WHERE PT = var_cfPT;

        commit;

END
```

visualizza_esercizi

Procedura che permette alla segreteria di visualizzare tutti gli esercizi nel Database.

```
CREATE PROCEDURE `visualizza_esercizi` ()
BEGIN
    SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
    START TRANSACTION;
                SELECT Nome
                FROM Esercizi;
    COMMIT;
END
```

visualizza_esercizi_mancanti

Procedura che permette ai clienti di visualizzare tutti gli esercizi che ancora non hanno effettuato durante la sessione di allenamento.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_esercizi_mancanti`(in var_cfCliente varchar(20),
        in var_dataSessione date,
        in var_dataInizioScheda date)
BEGIN

        SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
        start transaction;

                SELECT Posizione, Esercizio, NumeroSerie as Serie, NumeroRipetizioni as Ripetizioni
                FROM SchedaEsercizi se
                WHERE Cliente = var_cfCliente
                        AND dataInizioScheda = var_dataInizioScheda
                        AND Esercizio not in    (SELECT Esercizio
                                                 FROM EserciziSvolti es
                                                 WHERE es.Cliente = var_cfCliente
                                                         AND DataSessione = var_dataSessione
                                                         AND es.dataInizioScheda = var_dataInizioScheda)
                ORDER BY Posizione;

        commit;
END
```

visualizza_schede_attive

Procedura che permette ai clienti di visualizzare la loro scheda attiva.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_scheda_attiva`(in var_cfClienti varchar(20))
BEGIN
        declare var_Dat date;
    declare exit handler for sqlexception
        begin
                rollback;
                resignal;
        end;
        SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
        start transaction;

                SELECT DataInizio INTO var_Dat
                FROM Scheda
                WHERE DataFine IS NULL AND Cliente = var_cfClienti;

                IF var_Dat IS NULL THEN
```

```
                        SIGNAL SQLSTATE '45024' SET MESSAGE_TEXT = 'SCHEDA ATTIVA INESISTENTE';
                END IF;

                SELECT Posizione, Esercizio, NumeroSerie as Serie, NumeroRipetizioni as Ripetizioni
                FROM SchedaEsercizi
                WHERE Cliente = var_cfClienti and dataInizioScheda = var_Dat
                order by Posizione;


        commit;
END
```

visualizza_schede_archiviate

Procedura che permette ai clienti di visualizzare le data di inizio e di fine di tutte le loro schede archiviate.

```
 CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_schede_archiviate`(in var_cfClienti varchar(20), out var_numSchede int)
BEGIN
    declare exit handler for sqlexception
        begin
                rollback;
                resignal;
        end;
        SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
    start transaction;

        SELECT count(*) into var_numSchede
        FROM Scheda
        WHERE DataFine is not NULL and Cliente = var_cfClienti
                ORDER BY DataInizio ASC;

        IF(var_numSchede = 0) THEN
                        signal sqlstate '45025' set message_text = 'NON HAI SCHEDE ARCHIVIATE';
        END IF;

                SELECT DataInizio, DataFine
        FROM Scheda
        WHERE DataFine is not NULL and Cliente = var_cfClienti
                ORDER BY DataInizio ASC;

        commit ;

END
```

visualizza_schede_non_completate

Procedura che permette ad un Personal Trainer di visualizzare ogni cliente, nome, cognome e codice fiscale, che ha una scheda attiva non completa, insieme alla data di inizio della scheda.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_schede_non_completate`(IN var_PT VARCHAR(20))
BEGIN
	declare num int;

	declare exit handler for sqlexception
	begin
		rollback;
		resignal;
	end;

    SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
    START TRANSACTION;

		SELECT count(*) into num
		FROM Scheda s join Clienti on s.Cliente = CF
		WHERE PT = var_PT AND Tipo = 'non completata' AND DataFine is null;

		IF(num = 0) THEN
			signal sqlstate '45026' set message_text = 'NON HAI SCHEDE DA MODIFICARE, SONO TUTTE
COMPLETATE';
		END IF;

		SELECT Nome, Cognome, s.Cliente, s.DataInizio
		FROM Scheda s join Clienti on s.Cliente = CF
		WHERE PT = var_PT AND Tipo = 'non completata'
		ORDER BY DataInizio;

	COMMIT;
END
```

visualizza_serie_mancanti

Procedura che permette ad un cliente di visualizzare il numero di serie che gli mancano da fare per l'esercizio che sta eseguendo.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_serie_mancanti`(in var_cfCliente varchar(20),
	in var_nomeEsercizio varchar(30),
	in var_dataSessione date,
	in var_dataInizioScheda date)
BEGIN

	SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
	start transaction;
```

```
                SELECT es.Esercizio, (NumeroSerie-SerieCorrente) as SerieMancanti
                FROM EserciziSvolti es join SchedaEsercizi se ON es.Cliente = se.Cliente
                        AND es.dataInizioScheda = se.dataInizioScheda
                        AND es.Esercizio = se.Esercizio
                WHERE es.Cliente = var_cfCliente
                        AND es.dataInizioScheda = var_dataInizioScheda
            AND es.DataSessione = var_dataSessione
            AND es.Esercizio = var_nomeEsercizio;

        commit ;
END
```